**BEST PRACTICES**

**Application Development**

It is a good practice to structure a Python application by segregating functionality into multiple files (modules) and organizing them into folders (packages). This modular approach makes the codebase more maintainable, readable, and scalable. It also helps with debugging and allows for reusability of code.

**Typical Grouping and Organization**

Here’s how you could structure the files based on the functionality you’ve described:

**1. Root Directory**

The root directory should contain the main script to run the application and high-level files like requirements.txt, README.md, and configuration files.

**project/**

**│**

**├── main.py**

**├── requirements.txt**

**├── README.md**

**├── .env # Environment variables (if applicable)**

**├── config/**

**│ ├── settings.py # Configuration settings**

**│ └── logger.py # Centralized logging setup**

**2. utils/ (Utility Functions)**

A folder for common utilities that are reusable across the application.

* **File:** env\_utils.py
  + Functions for reading environment variables.
* **File:** db\_utils.py
  + Functions for establishing database connections.
* **File:** email\_utils.py
  + Functions for connecting to email and extracting attachments.

**3. features/ (Application Features)**

A folder to group functionality by major feature or task.

* **File:** file\_upload.py
  + Logic for uploading files to the database.
* **File:** file\_parser.py
  + Functions to parse file content and prepare data for database upload.

**4. data/ (Data Handling and Models)**

A folder for handling data-related logic.

* **File:** models.py
  + Data models or ORM models if using frameworks like SQLAlchemy.
* **File:** data\_validation.py
  + Functions for data validation and preprocessing.

**5. scripts/ (Automation or Batch Scripts)**

For standalone scripts like batch processing or scheduled tasks.

**project/**

**│**

**├── main.py # Entry point of the application**

**├── requirements.txt # Python dependencies**

**├── README.md # Project overview**

**├──.env # Environment variables**

**│**

**├── config/**

**│ ├── settings.py # Centralized settings/configuration**

**│ ├── config.ini # External configuration file (if needed)**

**│ └── logger.py # Logging configuration**

**│**

**├── logs/**

**│ └── logger\_output.log # Log output files**

**│**

**├── utils/**

**│ ├── env\_utils.py # Functions to handle environment variables**

**│ ├── db\_utils.py # Functions for database connections**

**│ └── email\_utils.py # Functions for email handling**

**│**

**├── features/**

**│ ├── file\_upload.py # Logic for uploading files to the database**

**│ ├── file\_parser.py # Parsing logic for data extraction**

**│ └── process\_files.py # Orchestrator for file-related workflows**

**│**

**├── data/**

**│ ├── models.py # Database schema definitions**

**│ └── data\_validation.py # Validation for file or extracted data**

**│**

**└── scripts/**

**└── batch\_job.py # Example batch script**

**Placement of Specific Files**

1. **config.ini:**
   * **Location:** config/ folder.
   * **Reason:** The config.ini file stores external configuration settings such as database credentials, API keys, and file paths, which align with other configuration logic in the config/ folder.

**Example: config.ini**

**[database]**

**host = localhost**

**user = admin**

**password = secret**

**name = mydb**

**[logging]**

**level = DEBUG**

**output\_file = logs/logger\_output.log**

1. **logger\_output.log:**

* **Location:** logs/ folder.
* **Reason:** Log files should be stored in a dedicated folder for better organization, easy access, and to separate them from code and configuration.

**Example: Logging Configuration in logger.py**

**import logging**

**# Create logger**

**logger = logging.getLogger("app\_logger")**

**logger.setLevel(logging.DEBUG)**

**# Create file handler**

**file\_handler = logging.FileHandler("logs/logger\_output.log")**

**file\_handler.setLevel(logging.DEBUG)**

**# Create formatter and add it to the handler**

**formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')**

**file\_handler.setFormatter(formatter)**

**# Add handler to logger**

**logger.addHandler(file\_handler)**

1. **Log Rotation (Optional):**

To prevent the logger\_output.log file from growing too large, you can use Python's logging.handlers.RotatingFileHandler.

**Example with Rotating Logs:**

**from logging.handlers import RotatingFileHandler**

**file\_handler = RotatingFileHandler("logs/logger\_output.log", maxBytes=5\*1024\*1024, backupCount=3)**

**Benefits of This Approach:**

* Separation of Concerns: Configuration files (config.ini) are kept in the config/ folder, while logs (logger\_output.log) are stored in logs/.
* Scalability: If the application grows, additional log files or configuration files can be easily organized in their respective folders.
* Clean Structure: Keeps the root directory clean and focuses on high-level files only.

This organization ensures your application is easy to navigate, scalable, and adheres to best practices for file placement and modularity.